REMARKS

Reconsideration and allowance of the present application are respectfully requested in view of the above-identified claim amendments in conjunction with the following remarks. Claims 20-24, 26, 32-35, 40, 49, 52, and 55 are currently pending in this application.

Regarding the Objection to the Declaration

The Office Action objected to the declaration because it makes reference to 37 CFR 1.56(a) rather than the entire rule, i.e., 37 CFR 1.56. It is submitted that the original declaration is sufficient, as § 1.56(a) establishes the operative part of the rule by stating, in part, that "Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section." A proper interpretation of materiality, as this term is used in section § 1.56(a), would implicitly incorporate the discussion of materiality in section § 1.56(b), although the declaration does not specifically mention this section. Section 1.56(c) states *inter alia* that each inventor has a duty of disclosure, which is implicitly established by the fact that each inventor has signed the declaration which makes reference to § 1.56(a). Therefore, the Applicant submits that the declaration, in its current form, satisfies the applicable rules.

Alternatively, the Applicant submits that the declaration's deviation from more preferable language is minor and can be waived under MPEP § 602.03 (as pointed out in the telephone interview of February 28, 2005). The Patent Office is respectfully requested to exercise its authority and waive the declaration's reference to § 1.56(a),

rather than § 1.56. This is considered appropriate because, as stated above, the declaration implicitly complies with all applicable rules.

35 U.S.C. § 103 Rejections

Claims 1-4, 13-15, 18-24, 26, 32-35, 41-43, 49, 51, 52, and 55 were rejected under 35 U.S.C. § 103(a) as being unpatentable over the combination of U.S. Patent No. 5,678,039 to Hinks et al. (referred to below as "Hinks") in view of "OpenWindows Developer's Guide: Xview Code Generator Programmer's Guide," by Sun Microsystems (referred to below as "Sun"), and further in view of U.S. Patent No. 6,802,059 to Lyapustina et al. (referred to below as "Lyapustina"). Claims 7, 8, and 10 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Sun in view of Lyapustina. Claim 9 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Sun in view of Lyapustina and U.S. Published Patent Application No. 2002/0107684 to Gao (referred to below as "Gao"). Claim 11 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Sun in view of Lyapustina and Hinks. And finally, claims 39 and 40 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Hinks in view of Sun, Lyapustina, and Gao. Applicant respectfully traverses each of these rejections for the reasons stated below. The rejections will be discussed with reference to the now-pending claims, i.e., claims 20-24, 26, 32-35, 40, 49, 52, and 55.

Consider first independent claim 20, reproduced below in full for convenience of reference (with emphasis):

20. A method comprising:

compiling a computer-servable document written for a particular locale ***to extract and remove characters associated with any original locale-sensitive content***, the compiling producing a compiled document with locale-independent elements;

storing the original locale-sensitive content;

substituting a function call in place of associated removed original locale-sensitive content in the compiled document; and

at runtime, retrieving the compiled document and populating the compiled document with a desired version of the original locale-sensitive content,

wherein the populating comprises executing the function call in the compiled document to obtain the desired version of the associated original locale-sensitive content and to insert the desired version of the associated original locale-sensitive content back into the compiled document.

The combination of Hinks, Sun, and Lyapustina was applied against this claim. The applied art is summarized as follows:

Hinks describes a technique for translating software into localized versions. With reference primarily to Fig. 3, Hinks' technique involves obtaining a resource file 325 from original source files 313 or from an original program 317. An EXPIMP (Export/Import Resource Parser) module 330 parses the resource file 325 into a Translation Table 340, which can be stored as a database table. The Translation Table 340 encapsulates all of the information that is known or can be derived from the various resources, and stores information in a format which may be utilized by various editors 350. Note generally column 7, line 53 to column 8, line 13 of Hinks, as well as Figs. 6B and 11, which illustrate the Translation Table 340. More specifically, certain information

in the Translation Table 340 is earmarked as translatable, and a user may use the editors 350 to translate such information. See, for instance, column 11, line 37 *et seq.* The translations can then be stored in the Translation Table 340. The EXPIMP module 340 can then produce a translated resource file 360, and the target product can be rebuilt based on this information. See generally column 8, lines 6-37.

Sun describes tools for internationalizing software programs. In the technique described by Sun, a Devguide tool or a developer inserts gettext() function calls around all user-visible text in an application. See page 99 of Sun, and also the samples on page 98. An xgettext operation can then be run on the source fields to produce a portable object file. See page 99 of Sun. The portable object file contains the native language strings taken from a program and placeholders for a localizer to supply each string's translation. See page 97. These portable object files can be shipped to localizers for translation into local languages. See page 98. The translated object files can be subsequently used to adapt a software program to another locale.

Lyapustina describes a conversion mechanism for inserting a particular body of text at various places within a program file. In one embodiment, the conversion mechanism performs a macro substitution by transforming hard coded strings into unique macro strings. To perform the macro substitution, the conversion mechanism is configured to receive a set of computer instructions that are contained in one or more files. In response to receiving the instructions, the conversion mechanism parses the instructions to identify character strings that have been included within the computer instructions. Upon identifying each string, the conversion mechanism generates a unique macro string as a substitute for the original string. The conversion mechanism then substitutes the unique macro string for the identified string in the source code of the computer program. The conversion mechanism also generates an entry in a macro list

that associates the unique macro string with the identified string for use during compilation of the computer instructions. For each file that includes instructions in which a unique macro string is substituted for an identified string, the conversion mechanism includes a reference to the macro list. At compilation time, a compiler may read the macro list and substitute each instance of the unique macro string with its associated identified string. See column 4, lines 7-31.

The applied documents, whether considered alone or in any combination, fail to render claim 20 obvious under 35 U.S.C. § 103(a). For example, none of the above documents describes at least the claimed elements of "compiling a computer-servable document written for a particular locale to extract and remove characters associated with any original locale-sensitive content," "substituting a function call in place of associated removed original locale-sensitive content in the compiled document," and "at runtime, retrieving the compiled document and populating the compiled document with a desired version of the original locale-sensitive content," "wherein the populating comprises executing the function call in the compiled document to obtain the desired version of the associated original locale-sensitive content and to insert the desired version of the associated original locale-sensitive content back into the compiled document."

For instance, Hinks does not substitute a function call in pace of associated removed locale-sensitive content. Namely, Hinks forms a Translation Table 340 from a program, translates any translatable content using editors 350, and then rejoins the translated content back to the program. The Examiner acknowledges the deficiencies of Hinks by stating that "Hinks does not expressly disclose the removal of locale-sensitive content or the substitution of a locale-sensitive content with a function call" (e.g., page 5 of the current Office Action, last paragraph).

The Office Action relies on the Sun and Lyapustina documents to make up for the shortcomings of Hinks. However, considering Sun first, this document uses function calls in a manner which is significantly different than the way the method of claim 20 uses function calls. In Sun, the gettext() function is added so that it *wraps around* existing locale-sensitive content. For example, if an application contained a message, "Please type your login ID," then the gettext() procedure might transform this statement as follows: gettext("Please type your login ID"). Then, after the gettext() function calls have been inserted around the text in a program, Sun runs a separate utility, the xgettext() operation, to create the portable file containing the locale-sensitive content. *But the message "Please type your login ID" remains intact in the application even after the xgettext() operation is run.*

First, Sun's adding of the gettext() function calls to the program cannot be said to *remove* the locale-sensitive content as claimed. Rather, the gettext() function calls *wrap around* the locale-sensitive content. At best, the gettext() function calls therefore can be said to *supplement* the locale-sensitive content, not remove it. Second, because the gettext() function calls do not remove the locale-sensitive content, Sun cannot be said to *substitute* a function call *in place* of associated removed locale-sensitive content. In other words, the gettext() function calls are not substitutes that stand in place of removed content, but rather simply mark content that remains present upon the addition of the gettext() function calls.

To further emphasize this point, claim 20 has been based on a comment that the Examiner made in paragraph No. 4 of page 3 of the Office Action. The relevant passage of the Office Action reads, "However, in the interest of advancement of the application, Applicant's present arguments have been interpreted more narrowly to require that a removal of locale-independent content would necessitate the removal of the original

sequence of characters present in Sun's text string." In accordance with this passage, claim 20 has been amended to recite extracting and removing "characters associated with any original locale-sensitive content." This makes it indisputably clear that content is physically removed from the document, not merely reinterpreted as part of a function call.

The Lyapustina reference is now added in the Office Action to address the removal and substitution aspect of the claimed invention. Namely, the Office Action states that "Lyapustina teaches that locale-sensitive content can be removed and substituted with a unique identifier string," citing column 4, lines 18-22 of Lyapustina for support (see page 6, last paragraph of the Office Action). Lyapustina describes substituting a unique macro string for an original string. However, the unique macro string serves as merely a reference that allows Lyapustina's conversion mechanism to insert a desired string in a program file at *compile time* (see, for instance column 7, lines 22-41 of Lyapustina). First, the unique macro string does not operate as a "function call" in the manner described above. Second, Lyapustina's populating occurs at compile time, whereas claim 20 recites "*at runtime*, retrieving the compiled document and populating the compiled document with a desired version of the original locale-sensitive content." In Lyapustina, the program file is presumably *already pre-populated* with the desired content at runtime.

As a final point, there is no motivation to combine the applied documents together, absent the use of impermissible hindsight. Consider first the combination of Hinks with Sun and Lyapustina. As summarized above, Hinks creates a Translation Table and then achieves transformation of this Table using various editors 350, such as a string editor, menu editor, dialog editor, and the like. These editors allow a human translator to easily access and manipulate the various resources of the program for

carrying out translation. See column 8, lines 6-13 of Hinks. Hinks regards this aspect its technique as a particular improvement over automatic string replacement algorithms. For example, Hinks states:

> Merely separating the text in a user interface from one's program is not an acceptable solution, however. Even after translating software prompts, help messages, and other textual information to the target languages, one still has to address basic issues of displaying and printing characters in the target language. (column 1, lines 30-35)
>
> . . . .
>
> At the level of application development, software translation is typically accomplished by extracting 'translatable strings' from various sources, translating them into the local language, and reinserting them back into the original sources for recompilation and linking. The approach has distinct disadvantages, however. For instance, the process of parsing different types of sources and extracting 'translatable items' from those sources is prone to error. Owing to the complexity of modern-day software development, sources are typically very diverse, ranging for example from strings embedded in C/C++ source to well-organized data sets in resource files. The ability of systems to reliably parse these different types of sources and identify each translatable item by a unique token is fairly limited. This limitation substantially affects the success of steps in the process and the reusability of tools from one project to another. (column 2, lines 30-44)
>
> To date, translation tools have focused on character-based information, taking into consideration only literal text strings. However, as more and more software development exploits graphical user interfaces, such as Microsoft Windows, this character-centric approach is inadequate. Other elements of the user interface require appropriate handling. For instance, the use of particular icons and bitmaps in one locale may be entirely

> inappropriate in another locale. Moreover, a system should take into consideration secondary effects of translation. For instance, translating a prompt in a dialog box from English to German may, in fact, require that the dialog box be resized (e.g., to accommodate a larger text string). (column 2, lines 45-57)

Sun and Lyapustina set forth techniques which pertain to the kind of automatic approach described by Hinks which involves "extracting 'translatable strings' from various sources, translating them into the local language, and reinserting them back into the original sources for recompilation and linking." As noted above, Hinks states that these approaches have "distinct disadvantages." Accordingly, a practitioner in the art who wanted to supplement Hinks in any manner would certainly not borrow from the techniques of Sun and Lyapustina, as these techniques are akin to the approach that Hink disparages. In other words, Hinks provides a technical platform that commends the use of various editors under the control of a human translator, rather than the automatic replacement strategies employed by Sun and Lyapustina. As such, one skilled in the art would not override the distinct preferences of Hinks by adding the kind of technology described in Sun and Lyapustina to Hink's invention.

Consider next the sub-combination of Sun with Lyapustina. Sun describes the use of gettext() and dgettext() routines for retrieving translated text (e.g., note page 98 of Sun). In contrast, as mentioned above, Lyapustina describes the use of unique macro strings in a program file which are used at *compile time* to perform string substitution, meaning that, at *runtime*, no substitution is performed. Sun and Lyapustina therefore describe two different techniques used in very different contexts. Hence, there would be no motivation to use the substitution mechanism described by Lyapustina in Sun's technique (even if complemented by Hinks). Namely, if Lypustina's technique is used

(in which strings are substituted at compile time), there is no longer any need for function calls which can be invoked at runtime. This means that these two techniques do not complement each other; rather, these techniques are mutually exclusively, such that if one technique is used, the other is not needed.

For at least the above-stated reasons, there is no motivation to combine Hinks, Sun, and Lyapustina. As stated in MPEP § 2143.01, if the proposed modification would render the prior art invention being modified unsatisfactory for its intended purpose, then there is no suggestion or motivation to make the proposed modification. *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984). Further, if the proposed modification or combination of the prior art would change the principle of operation of the prior art invention being modified, then the teachings of the references are not sufficient to render the claims prima facie obvious. *In re Ratti*, 270 F.2d 810, 123 USPQ 349 (CCPA 1959). The Applicant submits that adding Sun and/or Lyapustina to Hinks would run counter to the objectives of Hinks set forth in columns 1 and 2 of Hinks. Further, adding Lyapustina's substitution technique to Sun would entirely eviscerate the foundational principles of the gettext() routines described in Sun. As such, these kinds of contradictory combinations are specifically proscribed by the MPEP.

And for at least the above-stated reasons, Applicant also specifically traverses the various arguments purporting to establish the motivation for combining documents, for instance, as found in paragraph No. 6 of the Office Action and in the first paragraph on page 7 of the Office Action.

For completeness, the Applicant submits that the Gao reference does not make up for the deficiences of Hinks, Sun, and Lyapustina. Gao describes a technique for globalizing software. The technique parses software or website code to separate it into a file of international code (which is not locale dependent) and a resource pack of items

specific to a first locale. The internationalization process includes an analysis of the original code to identify its structure. Based on that analysis, the technique identifies potential items which contain information for insertion into the resource pack. See page 1, paragraph No. 5 to paragraph No. 7 of Gao. Since Gao does not describe the use of function calls in the manner recited in claim 20, Gao does not make up for the above-identified deficiencies of Hinks, Sun, and Lypustina.

For at least all of the above-stated reasons, the Applicant submits that none of the applied documents renders independent claim 20 obvious, whether these documents are considered individually or in any combination. The other pending independent claims (i.e., claims 32, 49, and 55) recite related features to claim 20, and are therefore allowable for reasons that are similar to those presented for claim 20. The dependent claims are allowable at least by virtue of their dependency on their respective independent claims. In addition, the dependent claims add subject matter which further distinguishes the invention recited in these claims over the applied art.

For at least the above-stated reasons, the Applicant respectfully requests the withdrawal of the 35 U.S.C. § 103(a) rejections.

New Claims

Claims 57-72 have been added. These claims depend variously from independent claims 20, 32, 49, and 55, and are allowable for at least this reason. In addition, these claims add subject matter which further distinguishes the invention recited in these claims over the applied art.

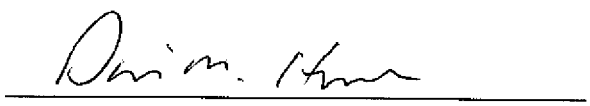
Conclusion

The arguments presented above are not exhaustive; Applicant reserves the right to present additional arguments to fortify its position. Further, Applicant reserves the right to challenge the prior art status of one or more documents cited in the Office Action, particularly the Gao reference.

All objections and rejections raised in the Office Action having been addressed, it is respectfully submitted that the present application is in condition for allowance and such allowance is respectfully solicited. The Examiner is urged to contact the undersigned if any issues remain unresolved by this Amendment.

Respectfully Submitted,

Dated: April 28, 2006

By: 

David M. Huntley
Reg. No. 40,309
(509) 324-9256